

# DATA PROJECT 1

Balma Agost, Borja Cabo, Inés Soler,  
Javier Ruiz y Luis Segura



# ÍNDICE

- CRITERIOS DE PUNTUACIÓN
  - ARQUITECTURA
  - MODELO DE DATOS
  - JUSTIFICACIÓN ARQUITECTURA

# **CRITERIOS DE PUNTUACIÓN**

**MÍNIMA PUNTUACIÓN POSIBLE:** 18 puntos  
**MÁXIMA PUNTUACIÓN POSIBLE:** 142 puntos

- Edad**
- Viudedad**
- Discapacidad**
- Acceso a transporte**
- Residencia de mayores**
- Comunidad Autónoma**
- Pensión**
- Programas anteriores**

# EDAD

- 1 65 - 75 años → 10 puntos
- 2 76 - 85 años → 15 puntos
- 3 > 85 años → 20 puntos



# DISCAPACIDAD



- 1    30% - 41% → 5 puntos
- 2    42% - 53% → 10 puntos
- 3    54% - 65% → 15 puntos
- 4    66% - 77% → 20 puntos
- 5    > 77%      → 25 puntos

# PENSIÓN



- 1 480€ - 996€ → 35 puntos
- 2 997€ - 1513€ → 20 puntos
- 3 1514€ - 2030€ → 10 puntos
- 4 2031€ - 2547€ → 5 puntos
- 5 > 2548 € → 1 punto

# PARTICIPACIÓN EN PROGRAMAS ANTERIORES



- 2022 & 2021 → 1 punto
- 2022 → 5 puntos
- 2021 → 10 puntos
- Ninguno → 25 puntos



# COMUNIDAD AUTÓNOMA

## 1 punto

- Comunidad Valenciana
- Murcia
- Andalucía

## 2 puntos

- Cataluña
- Galicia
- Asturias
- País Vasco

## 3 puntos

- Extremadura

## 4 puntos

- Madrid

## 5 puntos

- Aragón
- La Rioja
- Navarra
- Cantabria

## 6 puntos

- Castilla y León

## 7 puntos

- Baleares
- Canarias
- Ceuta
- Melilla

# BOOLEANOS

VIUDEDAD

10 puntos

RESIDENCIA DE MAYORES

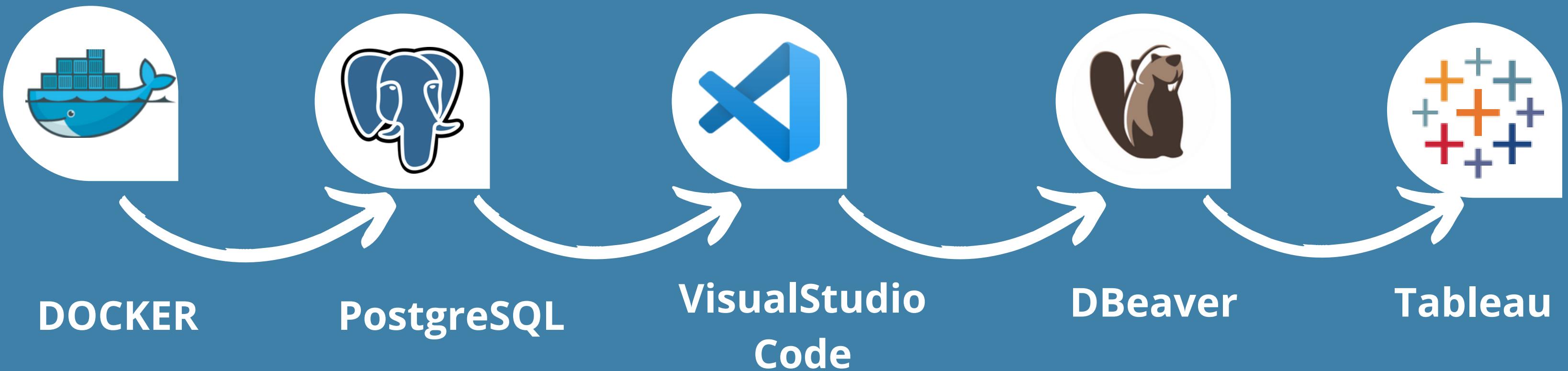
15 puntos

ACCESO A TRANSPORTE

1 ó 5 puntos



# ARQUITECTURA





# Dockerizando una aplicación Python

- Utilizamos la imagen base oficial de Python 3.9. proporcionando un entorno consistente y seguro para nuestra aplicación.

```
FROM python:3.9
```

- Establecemos el directorio de trabajo en el contenedor como /app.

```
WORKDIR /app
```

- Todas las operaciones siguientes se realizarán en el directorio establecido.

- Transferimos todos nuestros archivos de aplicación al directorio /app en el contenedor.

```
COPY . .
```

- Esto asegura que el contenedor tenga todos los archivos necesarios para ejecutar la aplicación.

```
RUN pip install --no-cache-dir -r requirements.txt
```

- Mediante requirements.txt, instalamos las dependencias necesarias.

- Establecemos el comando predeterminado para ejecutar la aplicación.

```
CMD ["bash", "-c", "python /app/fillments.py && python /app/programa.py"]
```



# Orquestación de Contenedores con Docker Compose

- Postgres:
  - Imagen: postgres:16.1
  - Configuración del Entorno:
    - Usuario:
    - Contraseña:
    - Base de Datos:
  - Puertos:
    - Mapeo del puerto 5432 del contenedor al puerto 5432 del host.
  - Volúmenes:
    - Utiliza un script SQL (tablas.sql) al iniciarse para configurar las tablas.
- Python:
  - Construcción de Imagen:
    - Contexto: Directorio actual (.)
    - Archivo Dockerfile:
  - Dependencias:
    - Depende del servicio postgres.
  - Volúmenes:
    - Permite sincronizar el directorio actual de la máquina local (.) con el directorio '/app' dentro del contenedor.

## Redes:

- Ambos contenedores están conectados a la red personalizada postgres.

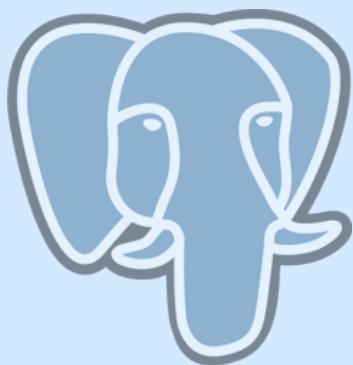
```
version: '3.5'

services:
  postgres:
    image: postgres:16.1
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: Welcome01
      POSTGRES_DB: data_project1
    ports:
      - "5432:5432"
    volumes:
      - ./tablas.sql:/docker-entrypoint-initdb.d/tablas.sql

  python-app:
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - postgres
    volumes:
      - .:/app

networks:
  postgres:
```

Permite la conexión eficiente entre los servicios PostgreSQL y Python.



# Definición de Tablas en Base de Datos

Mediante consultas SQL, se eliminan y crean tablas en nuestra base de datos, explicando cada estructura de tabla y sus relaciones, a través de claves primarias y foráneas.

```
DROP TABLE IF EXISTS public.solicitudes;
DROP TABLE IF EXISTS public.destinos;
DROP TABLE IF EXISTS public.ciudades;
DROP TABLE IF EXISTS public.preferencias;
DROP TABLE IF EXISTS public.hoteles;
DROP TABLE IF EXISTS public.puntuaciones;
DROP TABLE IF EXISTS public.asignaciones;
```

Se eliminan las tablas existentes, si las hay, para evitar conflictos al crear nuevas tablas.

```
CREATE TABLE public.solicitudes
(
    solicitud_id bigint PRIMARY KEY,
    nombre varchar(50),
    apellidos varchar(60),
    edad smallint,
    provincia_residente varchar(50),
    telefono varchar(50),
    discapacidad boolean,
    seguridad_social varchar(50),
    soltero_o_viudo boolean,
    vive_en_residencia boolean,
    viajara_con_acompanante boolean,
    acceso_transporte boolean,
    imserso_anopasado boolean,
    imserso_2021 boolean,
    importe_pension smallint,
    porcentaje_discapacidad smallint
);
```

```
CREATE TABLE public.puntuaciones
(
    solicitud_id bigint PRIMARY KEY,
    nombre varchar(50),
    apellidos varchar(60),
    puntaje smallint,
    FOREIGN KEY (solicitud_id) REFERENCES solicitudes(solicitud_id)
);
```

Creación de tablas en nuestra base de datos, entre las que se encuentran, las tablas “Solicitudes” o “Puntuaciones”, entre otras. Estableciendo claves primarias y foráneas.



- Definimos la función calcular\_puntaje, en la que iniciamos la variable “puntaje” en 0

```
def calcular_puntaje(datos):
    puntaje = 0
```

- Se especifican los criterios en los cuales vamos a basar nuestra función
- Creamos un nuevo DataFrame llamado puntuaciones\_df a partir del DataFrame original solicitudes\_df

```
puntuaciones_df = pd.DataFrame(solicitudes_df[['solicitud_id', 'nombre', 'apellidos']])
```

- Agregamos una nueva columna llamada 'puntaje' a este nuevo DataFrame.
- La función calcular\_puntaje se aplica a cada fila del DataFrame personas utilizando el método apply con axis=1.

```
puntuaciones_df['puntaje'] = solicitudes_df.apply(calcular_puntaje, axis=1)
```

- Establecemos la conexión e insertamos los datos en la tabla ‘Puntuaciones’
- Ordenamos el DataFrame por la columna 'puntaje' de mayor a menor

```
puntuaciones_df = puntuaciones_df.sort_values(by='puntaje', ascending=False)
```



- Definimos la función asignar\_hoteles

```
def asignar_hoteles(personas, hoteles_df, primer_recorrido=True):
```

- Agrupamos los hoteles por ciudad y sumando las plazas disponibles en cada ciudad. Esto se almacena en el DataFrame total\_plazas\_por\_provincia.

```
total_plazas_por_provincia = hoteles_df.groupby('ciudad')['plazas'].sum().reset_index()
```

- El DataFrame puntuaciones\_df se ordena en orden descendente según el puntaje de cada persona.
- **Bucle exterior o bucle de personas:**  
Obtiene las preferencias.  
Se extraen las preferencias de cada persona desde el DataFrame preferencias\_df.  
Iniciamos la lista vacía “hoteles asignados” para almacenar los nombres de los hoteles adjudicados a cada persona

```
for index, persona in puntuaciones_df.iterrows():
    # Obtener las preferencias de la persona
    preferencias = preferencias_df[preferencias_df['solicitud_id'] == persona['solicitud_id']][['opcion_1', 'opcion_2',
    ...
    hoteles_asignados = []
```



```
for preferencia in preferencias:  
    # Verificar si hay plazas disponibles en la ciudad de la preferencia  
    plazas_disponibles_provincia = total_plazas_por_provincia[total_plazas_por_provincia['ciudad'] == preferencia]['plazas'].values  
    if len(plazas_disponibles_provincia) > 0 and plazas_disponibles_provincia[0] > 0:  
        # Filtrar hoteles disponibles en la ciudad de la preferencia  
        hoteles_disponibles = hoteles_df[(hoteles_df['ciudad'] == preferencia) & (hoteles_df['plazas'] > 0)]  
  
        # Excluir hoteles que ya han sido asignados a esta persona  
        hoteles_disponibles = hoteles_disponibles[~hoteles_disponibles['hotel'].isin(hoteles_asignados)]
```

## Iteramos a través de las preferencias de la persona:

- Para cada preferencia de la persona, se verifica si hay plazas disponibles en la ciudad de la preferencia.
- Se verifica si hay plazas disponibles en la ciudad de la preferencia utilizando el DataFrame `total_plazas_por_provincia`.
- Si hay plazas disponibles en la ciudad, se procede a asignar hoteles
- Se excluyen los hoteles que ya han sido asignados a esta persona de la lista de hoteles disponibles.



```
for _ in range(2): # Intentar asignar hasta dos hoteles
    if not hoteles_disponibles.empty and len(hoteles_asignados) < 2:
        # Seleccionar el hotel con más plazas disponibles y asignar según el puntaje
        hotel_asignado = hoteles_disponibles.loc[hoteles_disponibles['plazas'].idxmax()]

        # Actualizar las plazas disponibles del hotel y ciudad
        plazas_a_restar = 2 if solicitudes_df['viajara_con_acompanante'].iloc[index] else 1
        hoteles_df.loc[hoteles_df['hotel_id'] == hotel_asignado['hotel_id'], 'plazas'] -= plazas_a_restar

        # Asignar hotel en el dataframe 'puntuaciones'
        puntuaciones_df.at[index, f'hotel_asignado_{_ + 1}'] = hotel_asignado['hotel']

        # Eliminar el hotel asignado de la lista de disponibles
        hoteles_disponibles = hoteles_disponibles[~(hoteles_disponibles['hotel_id'] == hotel_asignado['hotel_id'])]
        # Actualizar plazas disponibles en total_plazas_por_ciudad
        total_plazas_por_provincia.loc[total_plazas_por_provincia['ciudad'] == preferencia, 'plazas'] -= plazas_a_restar

        # Agregar el hotel a la lista de asignados
        hoteles_asignados.append(hotel_asignado['hotel'])
    else:
        break # Salir si no hay más hoteles disponibles
```

## Se inicia el bucle para intentar asignar hotel

- Intenta asignar hoteles siempre que haya disponibles y no se hayan asignado dos hoteles todavía
- Se selecciona el hotel con más plazas disponibles y se asigna según el puntaje de la persona.
- Se actualizan las plazas disponibles del hotel y la ciudad, y se ajusta la variable plazas\_a\_restar dependiendo de si la persona viajará con un acompañante.
- Se asigna el hotel en el DataFrame puntuaciones\_df.
- Se elimina el hotel asignado de la lista de hoteles disponibles
- Se actualiza la cantidad total de plazas disponibles en la ciudad.
- Se agrega el hotel asignado a la lista de hoteles asignados.

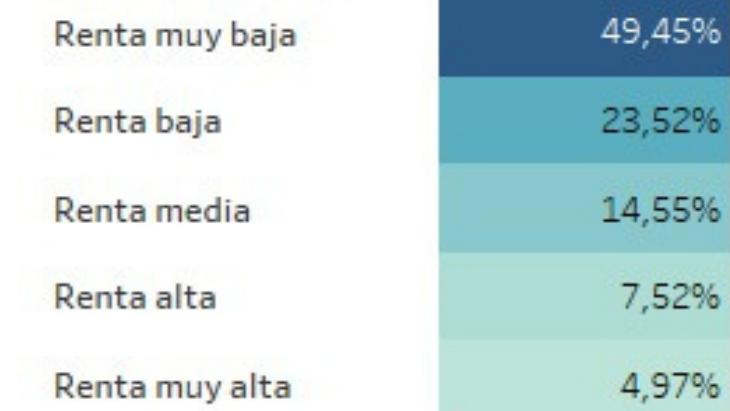


	123 solicitud_id	ABC nombre	ABC apellidos	123 puntaje	ABC hotel_asignado_1	ABC hotel_asignado_2
1	444 ↗	Máxima	Fabra	113	Exe Coruña	NaN
2	689 ↗	Álvaro	Sancho	112	Exe Oviedo Centro	NaN
3	727 ↗	Marino	Amaya	112	Pato Amarillo	NaN
4	375 ↗	Casemiro	Lamas	111	Vertice Chipiona Mar	NaN
5	197 ↗	Obdulia	Calleja	111	Htop Pineda Palace	NaN
6	1.491 ↗	Begoña	Bayón	111	Parasol Garden	NaN
7	1.863 ↗	Jorge	Izquierdo	108	Surf-Mar	NaN
8	257 ↗	Primitivo	Montero	108	Villa Luz	NaN
9	270 ↗	Arturo	Díez	107	Es Pla	NaN
10	1.861 ↗	Consuelo	Vizcaíno	107	Medplaya Balmoral	NaN
11	714 ↗	Martirio	Marti	106	Astoria	Blue Sea Costa Teguise Gardens
12	1.977 ↗	Jesusa	Merino	106	Tossa Beach Center	NaN
13	923 ↗	Casemiro	Donaire	106	Altamadores	Altamar
14	1.751 ↗	Caridad	Casares	106	Sancho Ramirez	NaN
15	706 ↗	Leopoldo	Revilla	103	Alua Parque San Antonio	Alua Tenerife
16	940 ↗	Cándido	Pagès	102	Port Feria	NaN
17	658 ↗	Teófilo	Boix	102	Equo Aranjuez	NaN
18	661 ↗	Cristian	Casas	102	Medplaya Bali	NaN
19	959 ↗	Plácido	Piña	102	Rey Sancho Ramírez	Albarracín
20	148 ↗	Zacarías	Dominguez	102	Gaudi	NaN
21	1.729 ↗	Fabiana	Ureña	102	Mora	Eurostars Zaragoza
22	533 ↗	Amanda	Coronado	102	Cap Negret	NaN

## Asignaciones de la opción preferida

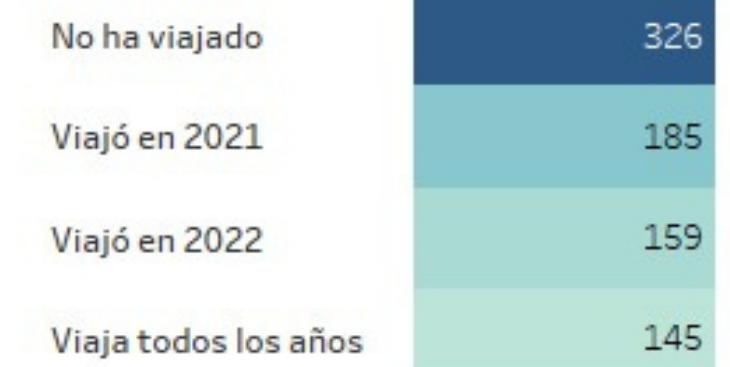
### Según la renta

#### Pensiones divididas



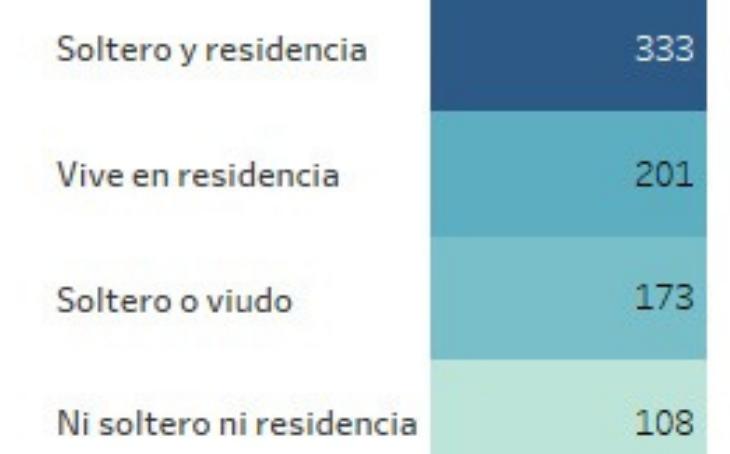
### Experiencias previas

#### Años de viajes



### Estado familiar

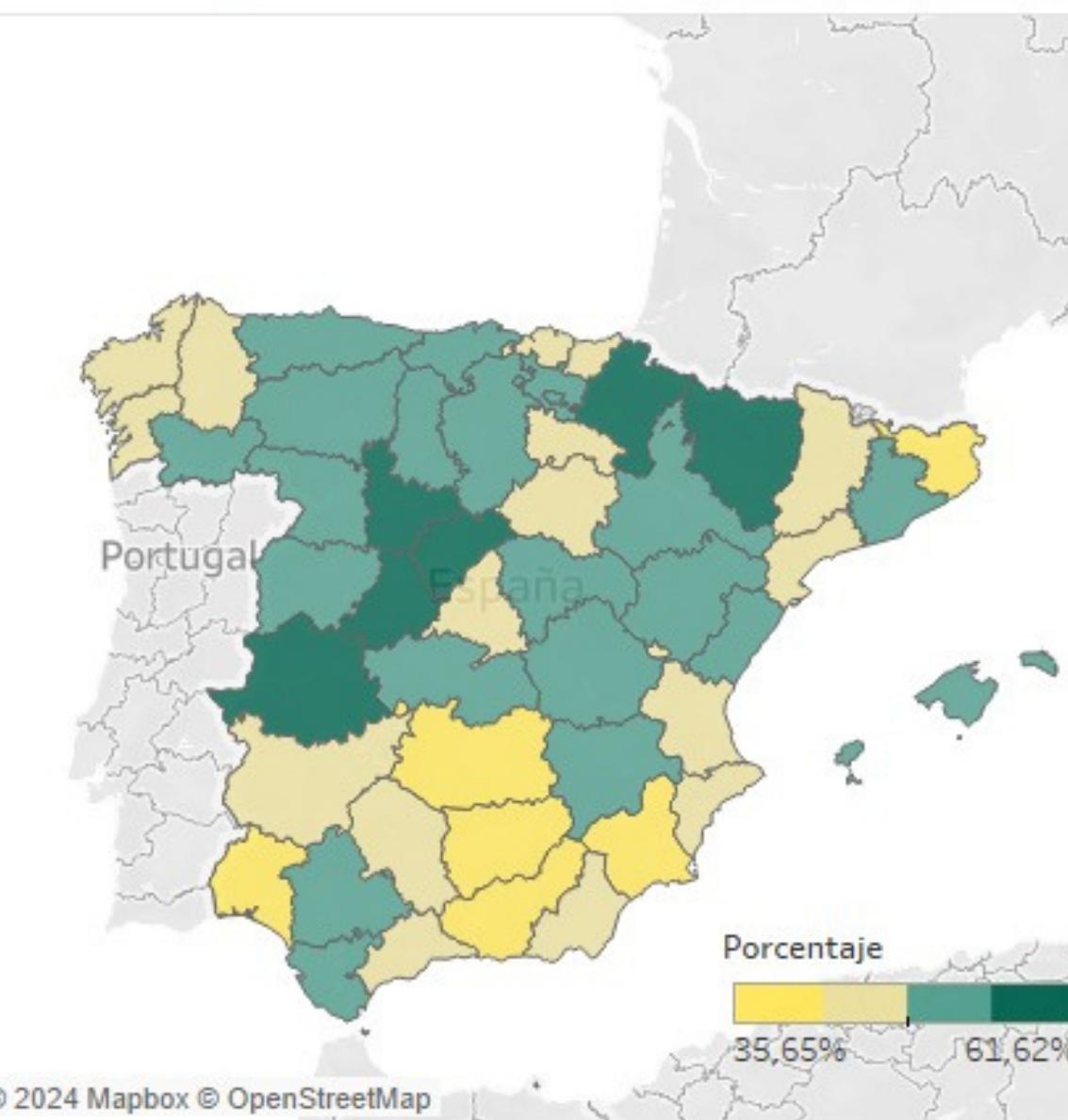
#### Soltero o Reside..



## Probabilidad de asignación según la edad



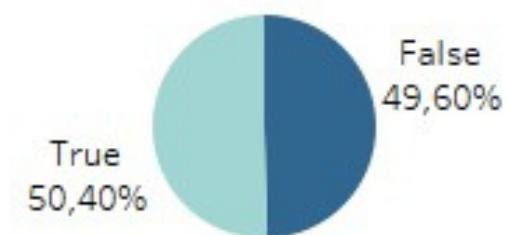
## Porcentaje de éxito en destino de preferencia



## Usuario promedio del Imserso

Prom. Importe Pension	1.251
Prom. Edad	82
Prom. Porcentaje Discapacidad	22
Prom. Puntaje	76

Porcentaje de discapacitados entre los usuarios



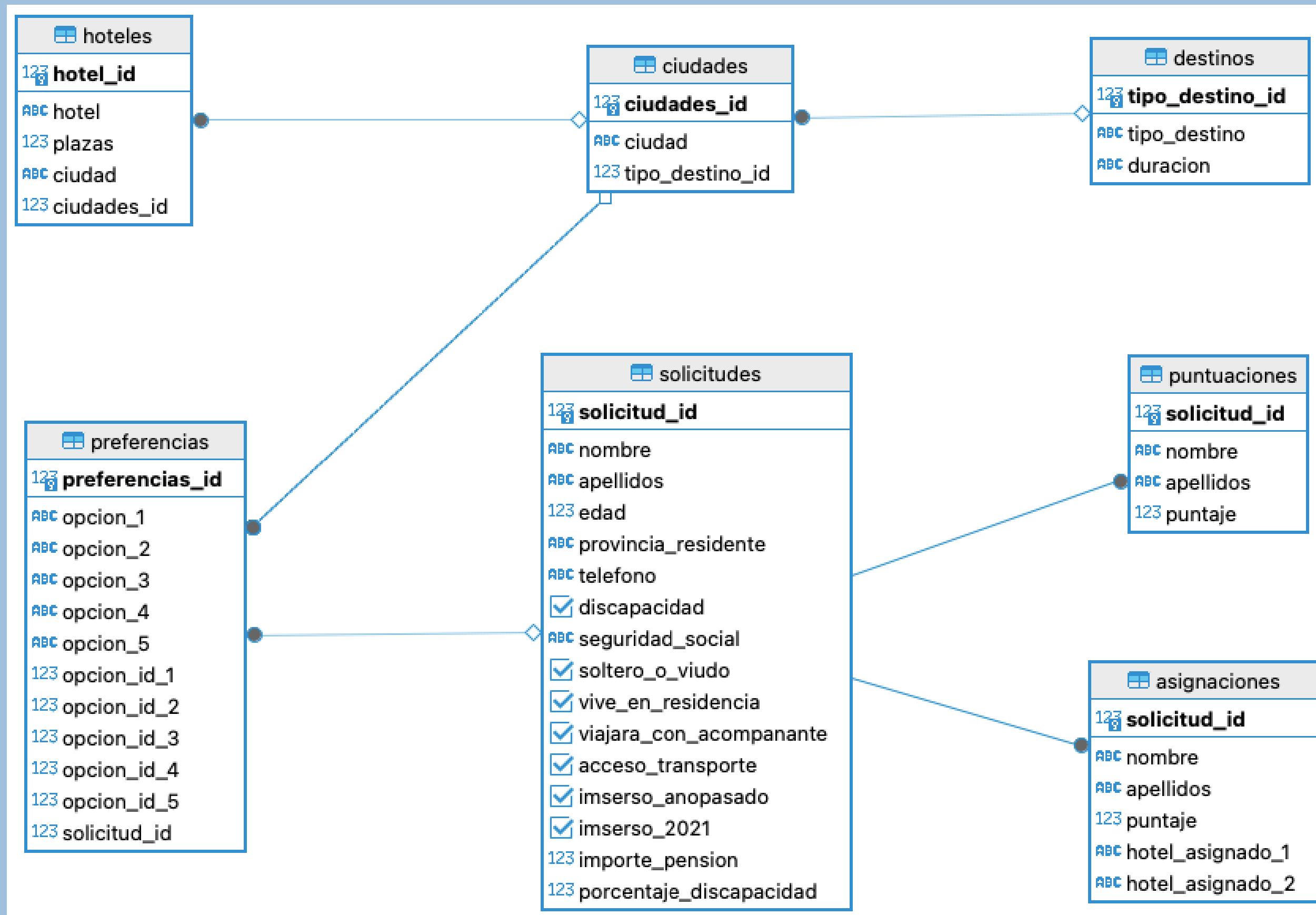
## Destinos más visitados



# **MODELO DE DATOS**



# Modelo Relacional



# Origen de los Datos

- Mayoritariamente creados de **forma aleatoria**: librería 'Faker' y módulo 'random' de Python.
- Datos creados aleatoriamente pero **basados en la realidad**:
  - Edad
  - Importe de las pensiones
  - Porcentaje de discapacidad
  - Plazas de hotel disponibles
- Datos extraídos de **fuentes reales**:
  - Posibles tipos de destino
  - Posibles ciudades en cada tipo de destino
  - Hoteles que participan en el programa del IMSERSO

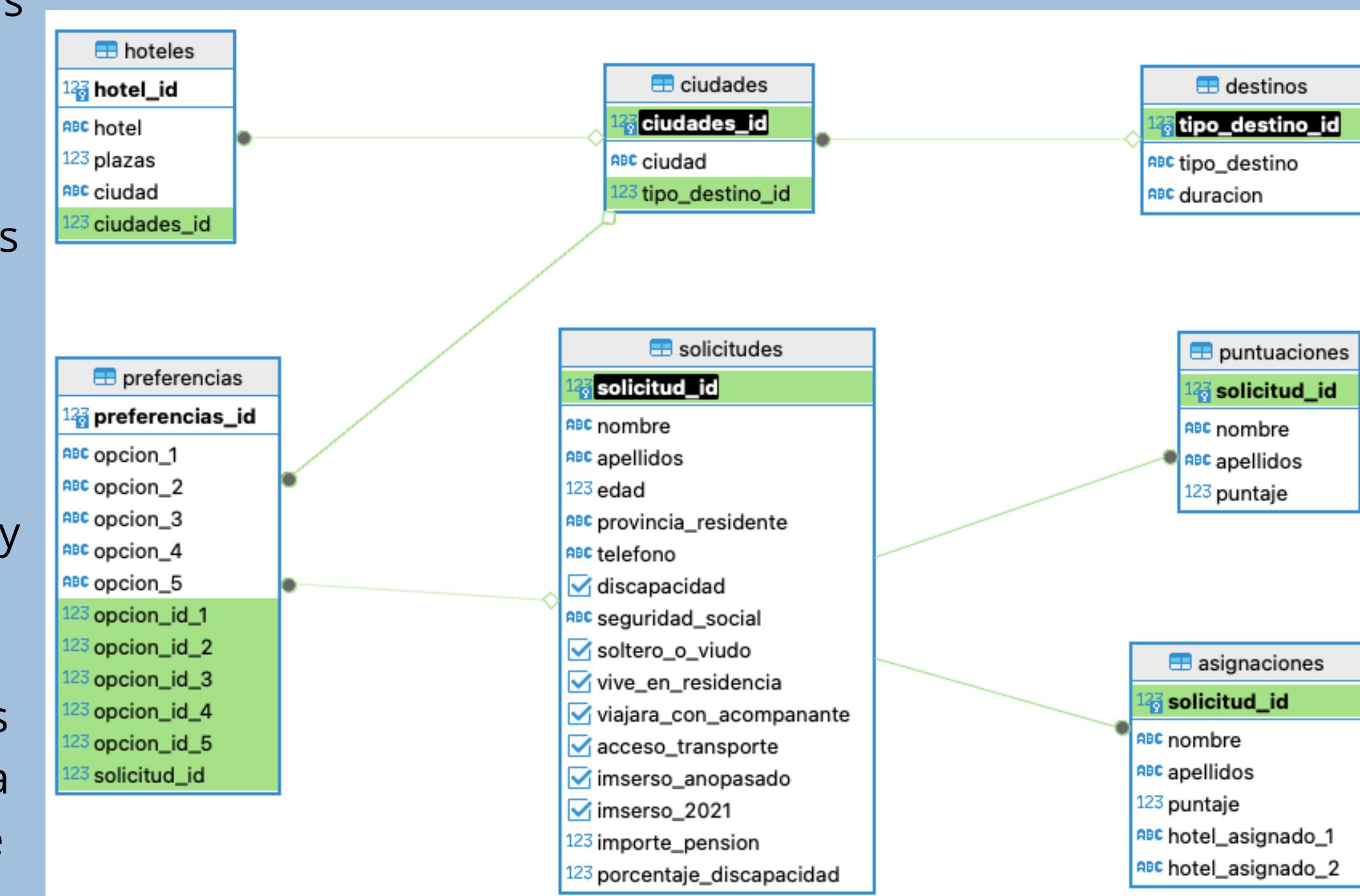
# Definiciones

En el modelado de datos, las "Definiciones" se refieren a las reglas y restricciones aplicadas a tablas y atributos para garantizar la integridad y consistencia de los datos.

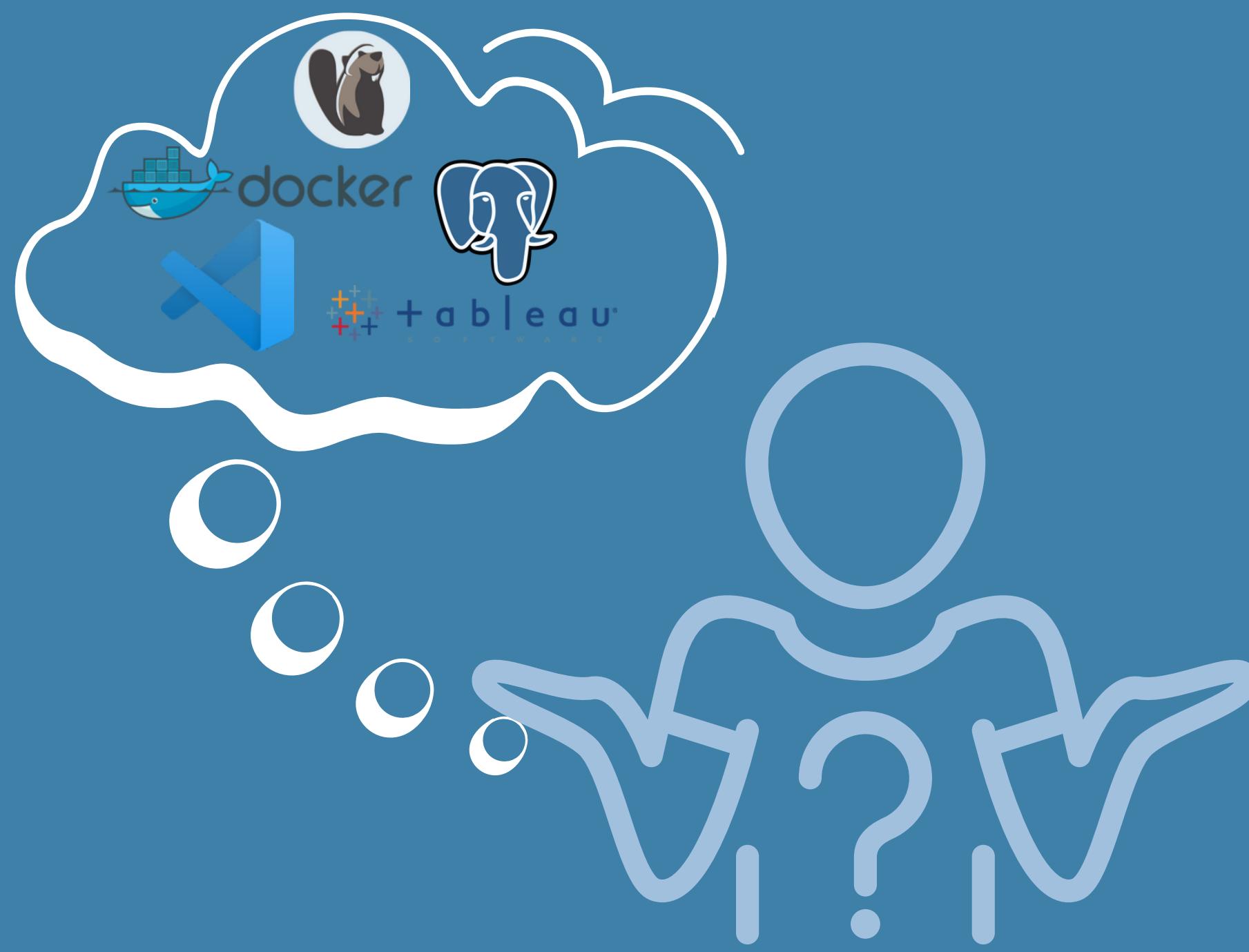
1. **Singularidad:** Asegura mediante claves primarias los duplicados, por ejemplo, "ON CONFLICT DO NOTHING", en consultas de inserción.

2. **Integridad referencial:** Se establecen relaciones entre las tablas para mantener la coherencia mediante claves foráneas para vincular demás variables relacionadas

3. **Restricciones de formato:** Controla el formato y el tipo de datos para cumplir con los criterios asignados: en este caso, los rangos de edad, pensión o preferencias en la asignación de hoteles y/o destinos. El uso de las mismas es esencial para mantener la consistencia del modelado de la base de datos.



# JUSTIFICACIÓN DE NUESTRA ARQUITECTURA ELEGIDA



# Docker

## Rápido inicio y detención

Los contenedores se inician y detienen rápidamente, lo que facilita el desarrollo y las pruebas ágiles.

## Portabilidad

Docker permite empaquetar una aplicación y todas sus dependencias en un contenedor, lo que facilita su despliegue en diferentes entornos sin preocuparse por las diferencias de configuración.

## Reproducibilidad

Docker utiliza el concepto de imágenes para garantizar la consistencia y reproducibilidad de los entornos de desarrollo, facilitando así el trabajo en equipo.



# PostgreSQL

## Almacenamiento eficiente y estructurado para datos relacionales.

Permite organizar la información en tablas con filas y columnas, lo que facilita la gestión y consulta de datos.

## Escalabilidad

PostgreSQL es capaz de manejar grandes volúmenes de datos y es escalable tanto en términos de tamaño de base de datos como de número de usuarios.

## Soporte para lenguaje SQL

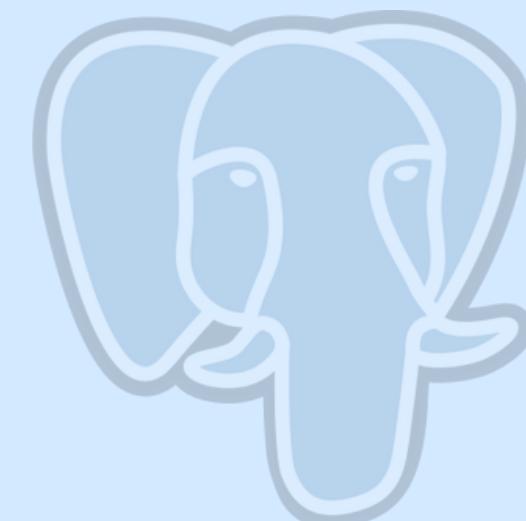
PostgreSQL utiliza el lenguaje SQL para realizar consultas y manipular datos.

## PostgreSQL proporciona opciones avanzadas para replicar datos

Los cambios realizados en una instancia de la base de datos se pueden propagar automáticamente a otras instancias de la base de datos.

## Integridad de datos

Puedes establecer claves primarias en una tabla (que garantizan unicidad) y claves foráneas que establecen relaciones entre tablas. Esto asegura que no se puedan realizar operaciones que violen la coherencia de los datos, como la eliminación de registros vinculados a través de una clave foránea.



# Visual Studio Code

## Familiaridad

Herramienta utilizada en gran parte del curso

## Integración con GitHub

Viene integrado con características de control de versiones Git, lo que facilita el seguimiento de cambios en el código.

## Herramientas de depuración integrada

Dispone de herramientas de depuración integradas son esenciales para simplificar el proceso de identificación y corrección de errores en el código durante el desarrollo.



# DBeaver

## Multiplataforma

Compatible con Windows, macOS y Linux

## Conectividad a múltiples bases de datos

Amplia gama de bases de datos, incluyendo MySQL, PostgreSQL, SQLite, Oracle, SQL Server, MongoDB...

## Interfaz intuitiva

Fácil uso

## Funciones avanzadas de edición y visualización

Permite a los usuarios editar y visualizar datos de manera efectiva.

## Herramientas de exportación e importación de datos

Permite la importación y exportación de datos de manera sencilla

## Múltiples herramientas

Proporciona herramientas integradas para el desarrollo SQL, como un editor de consultas SQL, resultado de sintaxis, sugerencias de código...



# Tableau

## Facilidad de uso

Tableau ofrece una interfaz intuitiva y fácil de usar.

## Potentes capacidades de visualización

Tableau ofrece una variedad de opciones de visualización, desde gráficos simples hasta cuadros de mando interactivos.

## Escalabilidad y despliegue en la nube

Puede adaptarse a las necesidades de grandes organizaciones de diferentes tamaños. Además, ofrece opciones de implementación en la nube.

## Conectividad a diversas fuentes de datos

Tableau es compatible con una amplia gama de fuentes de datos, incluyendo bases de datos relacionales, que es nuestro caso.

## Análisis en tiempo real

Tableau permite el análisis de datos en tiempo real, lo que significa que los cambios en los datos se reflejan automáticamente en las visualizaciones.

## Historial y seguimiento de cambios

Tableau ofrece la capacidad de realizar un seguimiento del historial de cambios en las visualizaciones y cuadros de mando, lo que facilita la colaboración en proyectos y la revisión de iteraciones anteriores.

# **i GRACIAS !**

**Enlace al repositorio de GitHub:** [https://github.com/inesssoler/Proyecto\\_imsero](https://github.com/inesssoler/Proyecto_imsero)

**Enlace al vídeo de Youtube:** [https://youtu.be/iL\\_1NI3e\\_xg](https://youtu.be/iL_1NI3e_xg)